Micro - lectures

without => 418 to 420 فاضل

# Lec.slide 1. Micro

What is Embedded system?

↳ Combination of hardware and software.

↳ Designed to Perform specific function.

↳ combination of computer hardware with the software to control, monitor, communicate to do multiple operations

ex: Washing machine, Digital Camera, DVD Player.

General Purpose system vs. Embedded system

| General Purpose | Embedded. |
|---|---|
| ↳ combination of generic HW & General Purpose OS for executing variety of apps. | ↳ combination of special Purpose HW and Embedded OS for executing specific set of apps. |
| Need not to be deterministic in execution behavior | Execution behavior is deter. for certain types of embedded system. |
| Example: Computer, laptop | Example: DVD, Printers. |
| | |

## Characteristics of Embedded systems:

* single functioned.
* Cost effectiveness
* fast execution time
* Reliability.
* low lower Consumption
* efficient use of memory.
* Processing Power.

## * Major application areas of Embedded System:-

1) Customer electronics (Cameras, etc)
2) Household appliances (TV, DVD Players, Washing machine)
3) Home automation & security system (fire alarms ... etc)
4) Telecom (handset, cellular telephones ... etc)
5) Healthcare (scanners, ECG, etc)
6) Banking & retail (ATM, Currency Counter, etc)

## Categorization of embedded system

[1] Based on Generalization:-

a. first generation (from 4 to 8 bit microcontrollers & ex: stepper motor control units)

b. 2nd Generation (8 to 16 bit micro Controllers & ex: sensors)

c. 3rd " (16 to 32 bit & ex: DSP )

d. 4th " (Smartphones, Mobile internet devices, etc)

[2] Based in Complexity & Performance :-

a. Small scale Embedded system

↳ small Processing elements with simple Program

↳ low cost. ↳ lower Performance.

b. Medium scale ES :-

↳ slightly complex in HW & SW.

↳ low cost. ↳ Medium performance.

c. Large scale ES

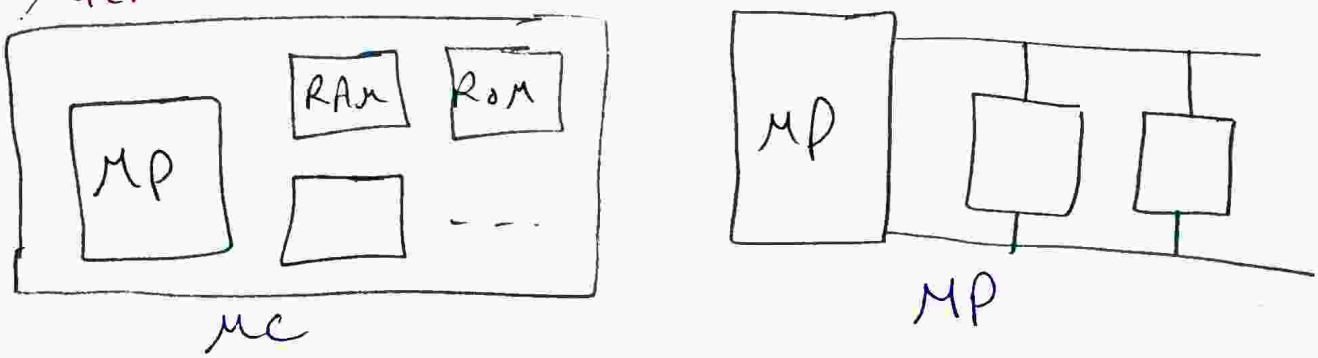
↳ complex ↳ High Performance. ↳ low cost

[3] Based on determinstic behavior.

↳ Hard real time ↳ soft real time.

[4] Based on Triggers (Event based & time based)

Micro controller vs Micro Processor




ES → uses (MC) to do specific task.

PC → uses MP to do different things.

* criteria of choosing MC

* speed * Power Consumption * Cost

* RAM & ROM * I/O Ports

* Packaging (Put MC in which Package)

* Community

Components of 8051 MC

a) 128 bytes RAM b) 4K ROM

c) 2 timers d) one serial port

e) 6 interrupt f) 4 I/O Ports (each 8 bits)

Notes

1) 8051 is subset of 8052

2) 8031 " a ROM-less 8051

| | 8051 | 8052 | 8031 |
|---|---|---|---|
| ROM | 4K | 8K | 0K |
| RAM | 128 | 256 | 128 |
| Timers | 2 | 3 | 2 |
| I/O Pins | 32 | 32 | 32 |
| Serial Port | 1 | 1 | 1 |
| Interrupt Sources | 6 | 8 | 6 |

4

3) * Vast majority of 8051 register are 8-bit registers.

## Micro Processor V-s Micro Controller

| Micro Processor | Micro controller |
|---|---|
| → use CISC Architecture | → use RISC and Harvard. |
| → has ROM, RAM, secondary storage memory, Ilo etc Placed on board and Connected through Buses. | → ROM, RAM, Ilo Prepherals are Combined in single integrated circuit |
| → less secured | → more secured. |
| → expensive | → cheaper |
| → Design takes more time | → Design it takes less time. |

| RISC | CISC |
|---|---|
| simple instruction format | Variable inst. format |
| large set of CPU registers | small set of general Purpose registers |
| very few addressing modes | large number of addressing modes. |
| simple Pipelinging | Complex Pipelining |
| supports on chip cache memory | seldom supports on chip cache memory |

# 8051 Assembly language Programming

1) How instructions are executed.

2) Registers.
   - a) 8 bit registers Me
   - b) A: Accumulator, must be destination of any operation.
   - c) B, PC, DPTR, SP
   - $R_{0-7}$

Any Assembly Program {instructions & directives}

instructions ⇒ [label:] opCode [operands] [; Comments]

Directives (not instructions)
- but they help assembler in assembly process.

ex ORG address ⇒ PC = address

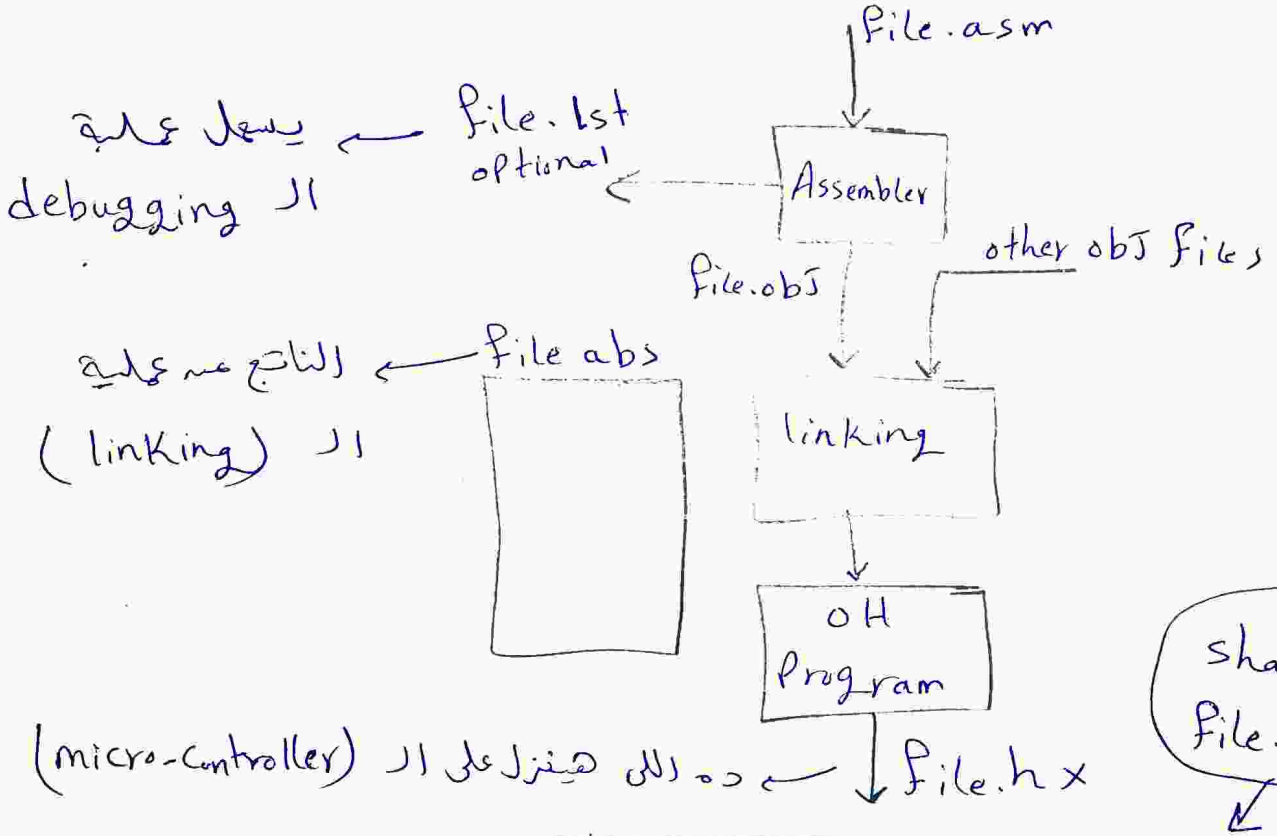
معناها إنه البرنامج بتاعي أول (instruction) هيبقى موجود في الـ (Address) اللي جنبه ده وهو اللي الـ (Program Counter) هيشاور عليه في الأول.

→ END

بيعرف الـ (Assembler) إنه دي نهاية الجزء اللي بيعمل ليه (Assembly).

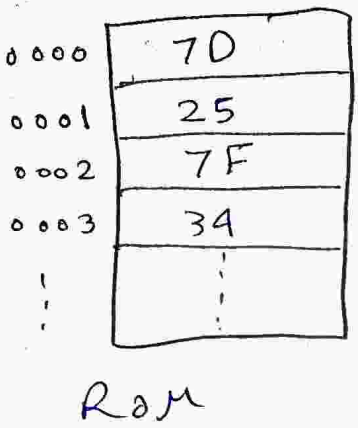
# step of assembly language Program

(file.asm) بكتب فيه ال (instruction) اللى عايزها
لـ بيدخل على ال (assembler) يطلع (file.object)
و و و التانيل اللى انتا محتاج يحصل ليه (linking) مع
الحاجة اللى انتا عايز تعملها .




| Address | Machine | Assembly | Comments |
|---|---|---|---|
| | | ORG 0000 | |
| 0000 | 7D25 | Mov R5, #25 | |
| 0002 | 7F34 | Mov R6, #34 | |
| 0004 | 2D | ADD A, R5 | |
| 0005 | — | — | |

أول ما تفتح الـ (Program) هتلاقي إنه
الـ PC عندك فيه PC=0000 وكل
شوية هيعمل (increment) وهيقرا عارف
إنه الـ (instruction) اللي جاي منين




ملحوظة: 1) مقدرتش أكتب برنامج السعة بتاعته أكبر من
4K byte ← ROM لأن سعة الـ (4K byte)

2) هيديلك برنامج (assembly) ويطلب منك تحط الجدول
بتاع الـ (lst.file) في الصفحة السابقة وبعدين تعمل الجدول في
أول الصفحة دي زي مثال في ص 74، 75 في (slides)

---

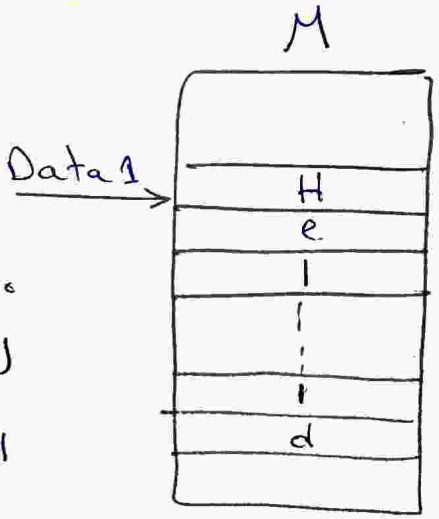
## DB directive

↳ most widely used ↳ used to define 8-bit data

شرح لو عايز أطبع (Hello world) بالـ (Assembly)
لـ محتاج تكون موجودة في الـ (memory)

```
ORG 500
Data 1 DB " Hello world"
```

لما عند الـ (lable Data1) اللي موجودة عند
500 كأني بقوله إنه بيشاور على الأمر دي.
لما أنا كده عرفت بالـ (DB) مجموعة الأوامر
الموجودة عند الـ (address 500)
محجوزة لـ (Hello world)



## *EQU

لما بعمل (define) لثابت مش غير مايأخد مكان في ال (memory) لما بعمل بيحط قيمة للثابت في (data label) وده لما ال (label) يظهر في البرنامج بيستخدم قيمة الثابت فوراً.

ex

COUNT EQU 25 → use EQU for Counter Constant

---

MOV R3, #COUNT → Constant used to load R3.

---

## Flag register (PSW) → Program status world

لغة عبارة عن (8-bit register) بيستخدم منهم 6-bit

| CY | AC | F0 | RS1 | RS0 | OV | -- | P |
|---|---|---|---|---|---|---|---|

(not used)

Where

CY → Carry. AC → Auxiliary Carry.

RS0, RS1 → Bank selectors. OV → overflow.

P → Parity flag. F0 → user definable.

ex

```
  1 1 1
  0 0 1 1 1 0 0 0      38
  0 0 1 0 1 1 1 1    + 2F
  ---------------    ----
  0 1 1 0  0 1 1 1    67
    6        7
```

عدد الوحايد
لو odd ← P=1
P=0 ← even

CY=0 ← 0=D7 عشان

انتقل من D3 ← D4 ومعاك

AC=1 ← Carry

P=1 , AC=1, CY=0

## * RAM : 128 bytes

(Variables) لـ مكان أخزن فيه الـ

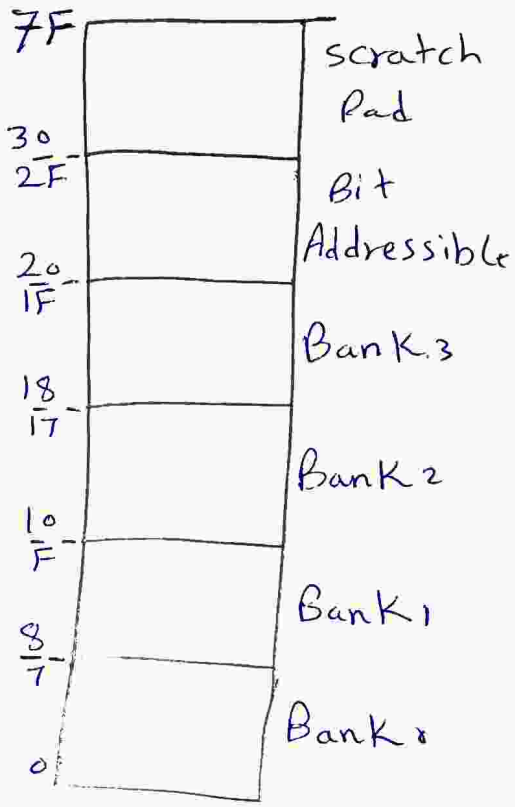
(Operations) اللي بعمل فيها

↳ (RS1, RS0)

(Bank) حسب قيمهم بختار الـ

اللي الـ (register) بتاعي بتعامل معاهم.

| RS0 | RS1 | Bank |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 2 |
| 1 | 1 | 3 |



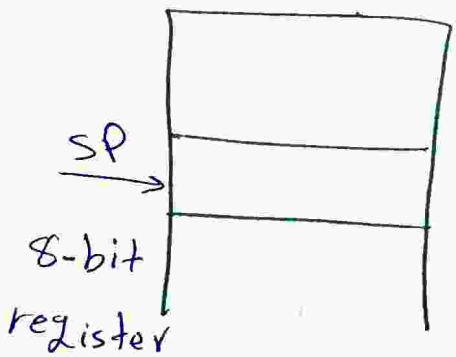

(stack) بتاعه هو (default) الـ (Bank 1)

↳ Push & Pop

## Default

SP = 7

عشان (Bank 1) تبدأ من (8)



Push → storing CPU register in stack
↳ increment then Push.

Pop → loading contents of stack back into CPU register.
↳ Pop then decrement.

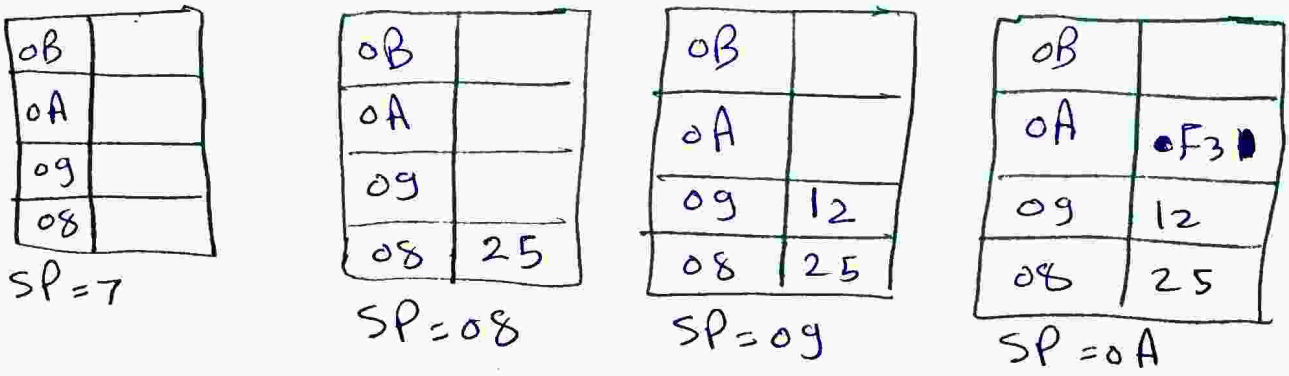
ملحوظة: لا يشترط ان (SP = 7) هو فقط بيشاور على عنوان معين وبيكون العنوان اللي بعده هو اللي بتخزن فيه الـ (data) بتاعتي.

---

Ex show stack and stack Pointer

```
Mov R6, #25H
Mov R1, #12H
Mov R4, #0F3H
Push 6
Push 1
Push 4
```

Sol



# JUMP, Loop and CALL instructions

**Loop** repeat sequence of instructions a certain number of times.

is performed by => DJNZ Reg, label

بعمل (decrement) لـ (reg) وبـ (check) كل شوية

لحالها ليس بوقف بعمل (Jump) لـ (label)

ex

Reg = value

— — —

— — —

DJNZ Reg, loop

بعمل (decrement) لـ (reg) وطول ماهو ليس بوقف بعمل (loop)

⇐ ماذا لو حتاج أكرر (action) أكتر من 256 مره

لأكبر قيمة لـ (reg) حيا (FF) 255 لكن لو عندي (loop) وتحتاج قيمة أكبر من (255) نعمل أكتر من (loop) جوه بعض (nested loop)

لو بقى مثلاً لو عندي 700 نعمل (2 loops) واحده 10 والأخرى 70، والناتج يكون حاصل ضرب بعض

# Jump

Conditional Jump:

لـ بعمل (operation) ثم أعمل (check flag) وعلى أساسه بعرف أخذ أى (action) فى البرنامج.

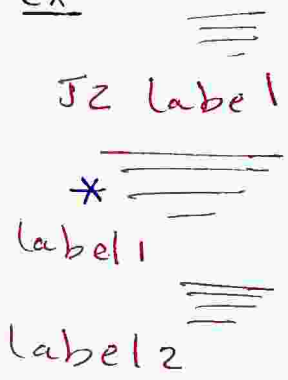
ex JZ label

لـ يعمل (check) على (zero flag) ومنها أعمل (Jump) لـ (label)

if equal to zero ⇒ Jump to label.

else (unconditional Jump) فى المنطقة * عمل (label 2) لـ

ex




## Jump

### 1) LJMP (long Jump)

لـ بعمل (Jump) لأى مكان فى البرنامج.

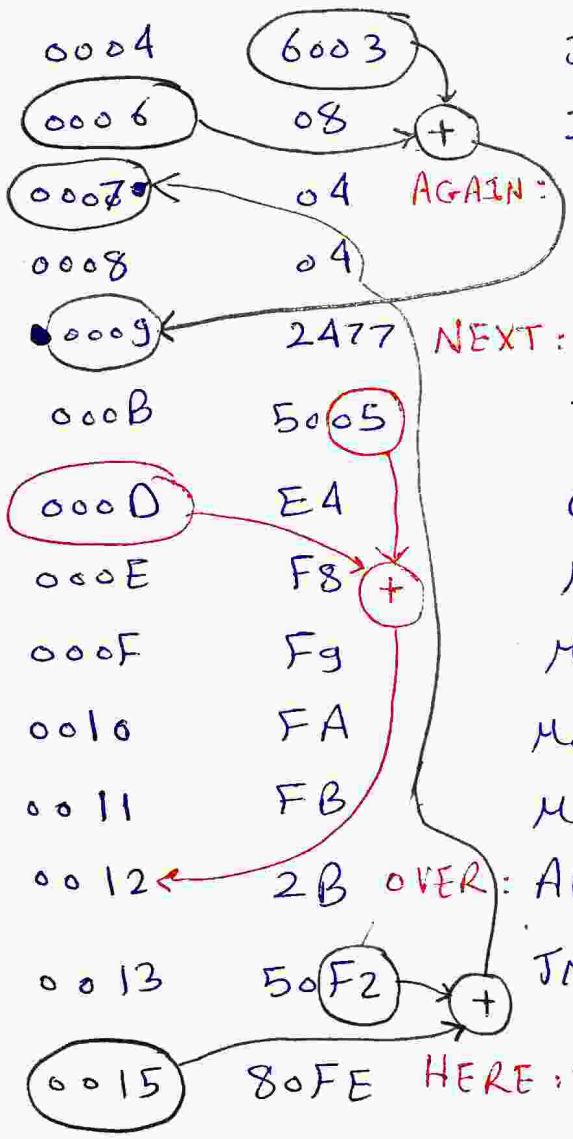
لـ لأنه بياخد (3 bytes) واحده لـ (opCode) , إتنين لـ (address)

### 2) SJMP (Short Jump) (-128 to 127)

لـ بقدر أعمل (Jump) فوق أو تحت بحد أقصى (128 bit) لـ بياخد (2-byte)

## Page 112 ex

| Line | Pc | opCode | operand |
|---|---|---|---|
| 01 | 0000 | | ORG 0000 |
| 02 | 0000 | 7800 | MOV R0, #0 |
| 03 | 0002 | 7455 | MOV A, #55H |
| 04 | 0004 | 6003 | JZ NEXT |
| 05 | 0006 | 08 | INC R0 |
| 06 | 0007 | 04 | AGAIN: INC A |
| 07 | 0008 | 04 | INC A |
| 08 | 0009 | 2477 | NEXT: Add A, #77H |
| 09 | 000B | 5005 | JNC OVER |
| 10 | 000D | E4 | CLR A |
| 11 | 000E | F8 | MOV R0, A |
| 12 | 000F | F9 | MOV R1, A |
| 13 | 0010 | FA | MOV R2, A |
| 14 | 0011 | FB | MOV R3, A |
| 15 | 0012 | 2B | OVER: ADD A, R3 |
| 16 | 0013 | 50F2 | JNC AGAIN |
| 17 | 0015 | 80FE | HERE: SJMP HERE |
| 18 | 0017 | | END |



ex

0004 60

0005 03

PC=0006 → + → 0009

هنا يجيب (opCode) لل Jump أو loop

عند JNC OVER

عايز يطلع بـ 5005

فبيطرح (0012) مكان ما الـ (OVER) راحت سه الـ (0D) الـ PC

كذلك مع JNC AGAIN

لو الـ (opCode) مكانه سه رقمين الـ PC هيزيد بـ (2)

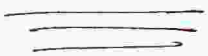
ولو رقم واحد الـ (PC) هيزيد (1)

# Call

- يكتب نوبة أكواد وفي النص عايز أعمل (subroutine) فبعمل ليه (Call)

ex

Call label1



sub1:

RET return

- الـ (Call) بيعمل Push لـ (Pc) ويعمل تغيير ليه لـ (sub 1) عشان يشاور عليه

Push PC

change Pc to sub 1

sub1 :

RET ⇒ Pop PC

- قبل الجزء بتاع (Sub 1) حطوا (HLT) معناها نهاية البرنامج

## Call

1) Long call (LCall)

- لما بعمل (Call) في أي مساحة في البرنامج (3-byte instruction) يعتبر
- الـ (subroutine) موجود في أي مكان خلال الـ (64K byte).

2) ACall (absolute call)

- لما بعمل (Call) في حدود (2K-byte) فقط (2-byte instruction) يعتبر

# Notes

لو في ال (main) وال (subroutine) بتستعمل نفس ال (registers) ⇐ ال (call) مش معنى بأنه يعمل ال Push لأنه بيتعامل مع ال (return ، Push) فقط.

لو في هذه الحالة قبل بداية ال (subroutine) هعمل Push لقيم ال (registers) اللي هستخدمها وفي الآخر هعمل (PoP) ليهم بس بالعكس.

Push 4
Push 5
PoP 5
PoP 4

لو عملت (Push) ومفيش في الآخر (PoP) مش هيطلع قيم ال (PC) الطلوبة.

لو لازم كل (Push) في البداية يقابلها نفس ال (PoP)

---

* ال (instruction) يتم تقريبة بال (crystal oscillator)

ال machine cycle : هي (cycle) يتم تنفيذ الأمر فيها

لو ال (microcontroller) يحتوي على مجموعة من ال (operations) كل (operation) تحتاج كام (cycle)

ال (machine cycle) لل (atmel) عبارة عن (12 clocks)

→ Find Period of machine cycle for 11.0592 MHz Crystal Frequency.

$$\frac{11.0592}{12} = 921.6\ KHz \quad \& \quad Machine\ cycle = \frac{1}{921.6KHz} = 1.085\ \mu s$$

# Addressing modes

1) Immediate
2) register.
3) Direct
4) register indirect
5) Indexed.

↳ Accessing memories.

## [1] Immediate

↳ src. operand is constant. MOV A, #25H

↳ load info. into any registers, including 16-bit DPTR register.

## [2] Register

↳ hold data to be manipulated (MOV A, R0)

↳ src. & dst. registers must match in size

ex MOV DPTR, A (XX) & MOV DPTR, #25F5H

↳ MOV R4, R7 (XX)

## [3] Direct addressing mode

↳ used to access RAM locations 30-7FH

ex: MOV A, 4 & MOV A, R4 (no # sign here)

* SFR (special function register)

↳ Can be accessed by their names or their addresses

↳ have addresses between 80H and FFH.

MOV 0E0H, #55H
MOV A, #55H
MOV 0F0H, R0
MOV B, R0

EX write code to send 55H to ports P1 & P2 using.

a) their names

```
Mov A, #55H
Mov P1, A
Mov P2, A
```

b) their addresses

```
Mov A, #55H
Mov 80H, A
Mov 0A0H, A
```

---

*stack and direct addressing mode

↳only direct addressing mode is allowed for pushing or popping the stack.

↳Push A (invalid) ⇒ it should be Push 0E0H

EX show the code to Push R5 and A onto the stack and then pop them back into R2 & B, (B=A, R2=R5)

sol

```
Push 05        ; Push R5 to stack
Push 0E0H      ; Push register A to "
Pop  0F0H      ; Pop top of stack to B
                 now B = A
Pop  02        ; Pop top of stack into R2
                 now R2 = R5
```

## 4 register indirect addressing mode :-

↳ register is used as a pointer to data (only R0, R1)

MOV A, @R0 → move contents of RAM whose address is held by R0 into A

MOV @R1, B

EX write a program to copy value 55H into RAM memory locations 40H to 41H using :

a) direct addressing mode

```
MOV A, #55H
MOV 40H, A
MOV 41H, A
```

b) register indirect without loop

```
MOV A, #55H
MOV R0, #40H
MOV @R0, A
INC R0
MOV @R0, A
```

c) register indirect with loop

```
    MOV A, #55H
    MOV R0, #40H
    MOV R2, #02
AGAIN: MOV @R0, A
    INC R0
    DJNZ R2, AGAIN
```

↳ advantage of register indirect that it makes accessing data dynamic rather than static as in direct addressing mode.